

ISP RAS Projects on

Improving GCC for Itanium

Arutyun Avetisyan (arut@ispras.ru)
Andrey Belevantsev (abel@ispras.ru)

Gelato Itanium Conference and Expo
April 16th, 2007
San Jose, CA

Agenda

- Introduction to ISP RAS work on GCC
- Support for control and data speculation
- Selective scheduling and pipelining (2006-07)
 - Basic features
 - Implementation issues
 - Preliminary evaluation
- Propagating aliasing information from Tree SSA to RTL (2005-06) /
Further work on improving RTL aliasing (2007)
- Fixing modulo scheduling for Itanium /
Propagating data dependency info to RTL (2007)

Introduction to GCC work on Itanium

- Itanium platform relies on a compiler for achieving good performance
 - And provides a lot of resources and features for that
- Instruction scheduling, software pipelining, and memory disambiguation are especially important
- GCC has problems in all these areas
 - Instruction scheduling is not aggressive enough
 - Swing modulo scheduling does not work on Itanium
 - Alias analysis on lower IR level (RTL) is weak
- ISP RAS has a number of projects to address these issues

Previous meetings on Itanium GCC

- 1st Gelato GCC workshop in Geneva, Jan 2005
- GCC track on Gelato ICE San Jose, Apr 2006
- 2nd Gelato GCC workshop in Moscow, Aug 2006
 - Organized by Gelato and ISP RAS
 - <http://gcc.gelato.org/MoscowMeeting>
- High-priority projects agreed upon
 - Alias analysis
 - Instruction scheduling
 - Software pipelining
 - Prefetching
 - Link-time optimizations
 - Feedback-directed optimizations

Control/data speculation

- Implemented and tuned in 2005
- Checked into mainline in March 2006
- Since then:
 - 9 bug reports were fixed -
thanks to Martin Michlmayr for finding them!
 - new scheduler dependence lists (for PR28071)
 - various cleanups to the scheduler
- Will be in GCC 4.2.0 (coming soon)
- The default tuning (-O2) is quite conservative
 - More aggressive behavior could be requested via flags
 - Better instruction priorities are needed
 - A project for Google Summer of Code is planned for this

Selective scheduling and pipelining

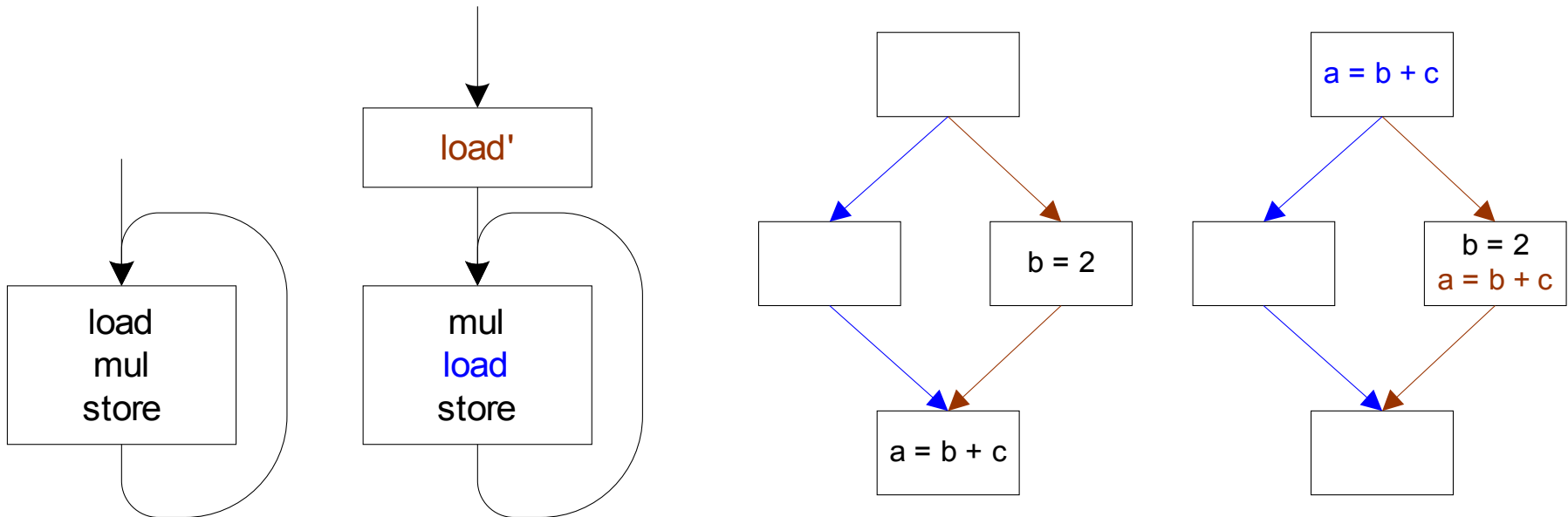
- New scheduling framework for GCC
 - Based on the selective scheduling (focuses on VLIW)
 - Supports scheduling along all paths in a DAG
 - Supports a number of instruction transformations
 - instruction cloning, speculative code motion, multiway branching, register renaming, forward substitution
 - Could be easily extended and tuned
- Provides software pipelining implementation
 - with a support for control-flow intensive and non-countable loops
 - can pipeline loop nests starting from innermost loop to the outermost
 - supports control/data speculation during pipelining

The project summary

- Requires a lot of development efforts
 - 15 months for basic implementation
 - 9 months for performance tuning
 - One of the biggest projects in the GCC backend
- Has a team of five
 - Consulting is provided by Vladimir Makarov, Red Hat
- Started in September 2005
 - Sources are published in January 2007
- Available in “sel-sched-branch” branch of GCC
 - `svn co svn://gcc.gnu.org/svn/gcc/branches/
sel-sched-branch`
- Aimed for inclusion in GCC 4.4

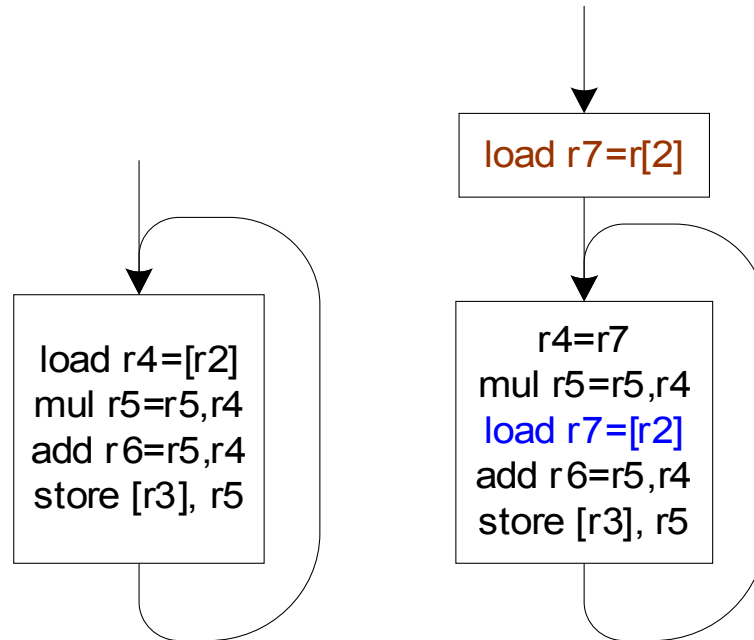
Features highlight

- Instruction cloning / Bookkeeping code support
 - Useful for moving insns to blocks that do not dominate the source block – e.g., pipelining
 - Useful when an insn is not available along all paths



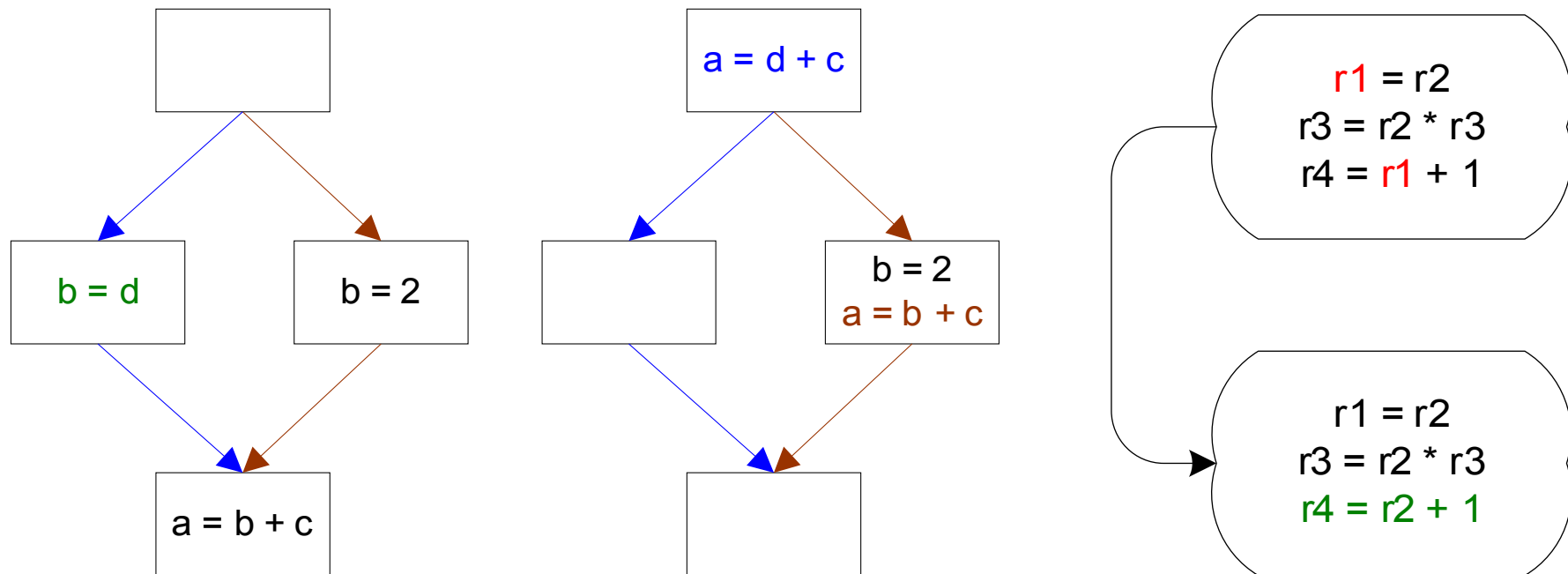
Features highlight

- Register renaming support
 - Useful for eliminating spurious true dependencies
 - Enhances pipelining opportunities



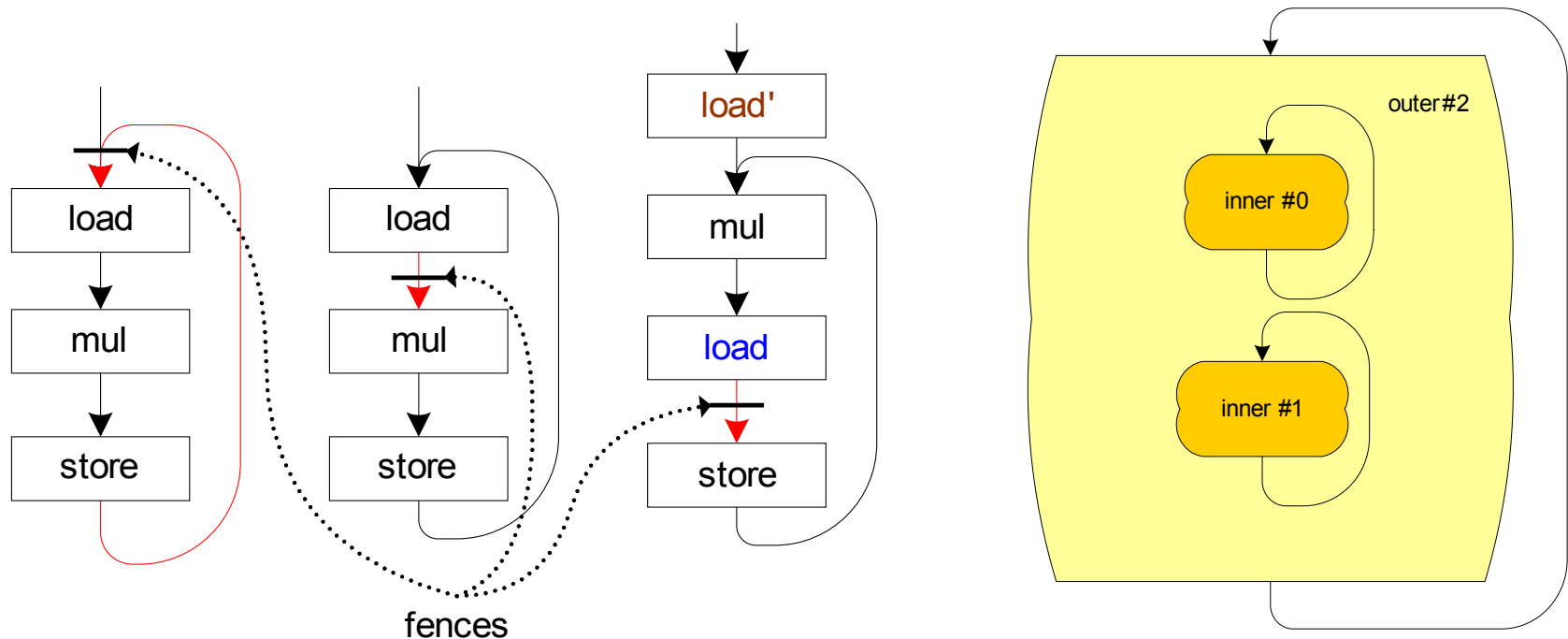
Features highlight

- Forward substitution support
 - Propagates insns through “reg1=reg2” copies
 - Can do this also inside an instruction group

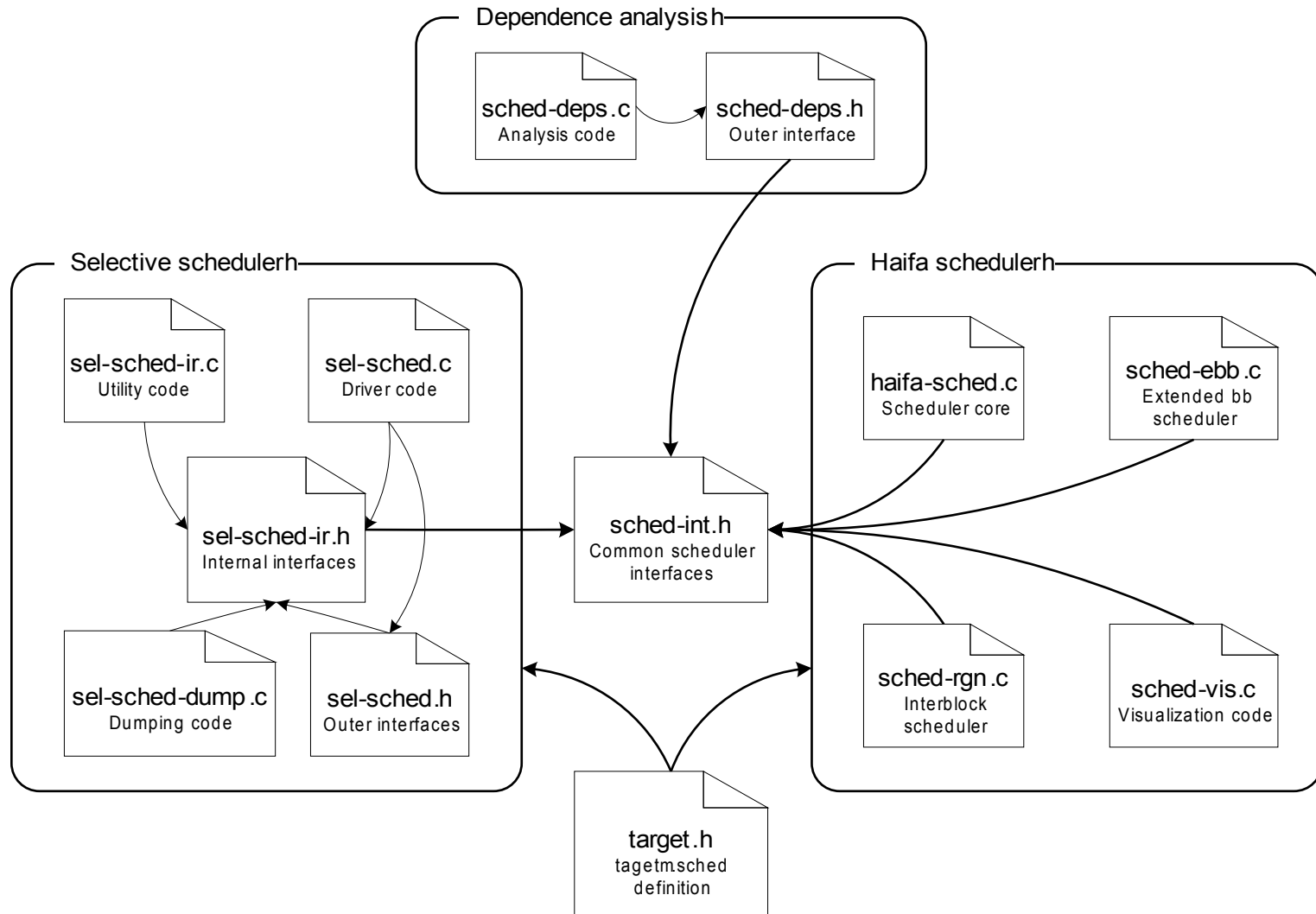


Features highlight

- Software pipelining support
 - Pipeline innermost loops via “dynamic” back edge
 - A *fence* serves as a barrier for code motion
 - Pipeline loop nests starting from innermost loops
 - Treat inner loops like barriers



Anatomy of GCC schedulers



Implementation highlights

- A flexible dependence analyzer framework
- A framework for initializing scheduling data
 - Initialize new insns
 - Initialize new basic blocks
 - Add new regions
- Choosing best instruction for scheduling
- Support multiple scheduling points (fences)
- Improving the IA-64 backend
 - Alignment issues
 - Placement of stop bits
 - Instruction grouping heuristics
- Code sharing between the schedulers

Dependence analyzer

- A callback interface to analyzer internals
 - {start, finish}_ {insn, lhs, rhs}
 - note_reg_ {set, use, clobber}
 - note_dep, note_mem_dep
- Callbacks are used for various purposes
 - Global init of insn flags (SCHED_GROUP_P)
 - Computation of availability sets (moveup_rhs)
 - Checking that insn's input operands are ready
 - The Haifa scheduler has its own callbacks
- A client should implement its own caching if non-standard callbacks are used
 - A work to do this in selective scheduling is underway

Initializing scheduling data

- New instructions are initialized through a single entry point
 - Set initialization flags
 - Scan new insns/bbs calling user-defined hooks for each (global init, extra jumps, bb headers, finalization)
- Jumps generated when modifying control flow are caught with an RTL hook
- New basic blocks are handled inside wrappers for standard CFG functions
 - Blocks are added to the current region
 - “Unexpected” basic blocks are caught with a CFG hook
- New regions are created
 - For recovery blocks and for loop preheader blocks

Choosing best instruction

- DFA lookahead engine is adapted
 - A ready list is generated from the final availability set
 - We feed it to `max_issue ()`
- Scheduler hooks are called in proper places
 - `reorder/reorder2` hooks use the generated ready list
 - `dfa_new_cycle/variable_issue`
- Instruction priorities are calculated
 - Using dependencies constructed during global init
- Better priorities are needed to drive pipelining/speculation
 - Can reuse results from the Google SoC application

Supporting multiple scheduling points

- A fence represents the current scheduling point
- Some previously global data should be encapsulated inside fences
 - Current DFA state
 - Dependence contexts (check whether an insn is ready)
 - *Target contexts*
 - Opaque type hiding machine-dependent structure
 - Allows a target to reflect the correct scheduler state when inside a scheduler hook
 - Itanium backend uses this for determining register conflicts inside an insn group and placing stop bits
 - Various flags
 - Last scheduled insn, current cycle, did we stall, etc. etc.

IA-64 backend improvements

- Alignment issues
 - Functions should be aligned to 64 bytes
 - Targets of loop branches should be aligned to 32 bytes
 - Additional care should be taken for the (already) aligned loop label to stay with the loop
- Placing of stop bits
 - It is desirable to put a stop bit after every cycle to avoid stalling the whole instruction group

<code>fma f10=...</code>	<code>//4 cycle latency</code>	<code>fma f10=...</code>	
<code>ldf f11=[...] ;;</code>	<code>// 6 cycle latency</code>	<code>ldf f11=[...] ;;</code>	
<code>fma = f10,...</code>	<code>//Both fmas in</code>	<code>fma = f10,...</code> ;;	<code>//Starts earlier</code>
<code>fma = f11 ;;</code>	<code>// same insn group</code>	<code>fma = f11 ;;</code>	<code>//Waits for ldf</code>

- Target contexts support

Code sharing between the schedulers

- Region formation routines
 - But we use loop optimizer for pipelining of outer loops
- Dependence analyzer
 - Different hooks are needed
- DFA engine
 - We adapt to Haifa's interface
- Scheduling hooks
 - No machine-dependent special casing
- Instruction priorities
- Parts of speculation support
 - Recovery blocks
 - Target support

Current status

- All basic infrastructure is implemented
 - Bookkeeping support, register renaming, forward substitution
 - Software pipelining (innermost loops and loop nests)
 - Data and control speculation
 - Can be run before and after register allocation
- We are concentrated on performance tuning
 - Testing on SPEC CPU 2000 benchmarks
 - Better instruction priorities
 - Using of call saved registers for renaming
 - Rescheduling of pipelined loops for tighter schedules
 - Heuristics controlling aggressiveness of pipelining
 - Further tuning of the IA64 backend

Results – small benchmarks

Benchmark	-O2	-O3 -funroll-loops	Benchmark	-O2	-O3 -funroll-loops	
Whetstone	-0,50%	-1,94%	FLOPS	7,15%	3,46%	
Tfftdp	1,62%	-1,65%		5,96%	8,82%	
Sim	1,16%	0,00%		7,05%	9,81%	
Shuffle	0,92%	0,68%		10,46%	12,57%	
Linpackc	6,72%	3,89%	Hanoi: 29 disks	0,06%	0,04%	
Heapsort	0,00%	5,17%	Fibonacci	-0,14%	-0,99%	
Dhrystone 1.1	-1,51%	-1,19%	Queens	24,04%	6,85%	
Dhrystone 2.1a	1,27%	2,28%	Scimark	0,00%	6,93%	
c4 : Fhourstones 1.0	5,10%	-0,61%		FFT	2,61%	1,85%
Black Jack	0,95%	0,66%		SOR	1,55%	7,07%
Nsieve size=8191	9,71%	8,16%		Monte Carlo	0,04%	0,04%
FFT size=1000000	1,05%	0,13%		Sparse Matmult	-3,18%	8,70%
Blowfish	0,43%	-0,09%		LU	0,03%	9,17%

Speedup % to the old scheduler - no speculation support in both schedulers

Results – SPEC

168.wupwise	500	511	2,20%	168.wupwise	501	523	4,39%
171.swim	712	732	2,81%	171.swim	733	770	5,05%
172.mgrid	322	372	15,53%	172.mgrid	332	373	12,35%
173.applu	436	432	-0,92%	173.applu	438	447	2,05%
177.mesa	744	739	-0,67%	177.mesa	726	747	2,89%
178.galgel	691	705	2,03%	178.galgel	686	814	18,66%
179.art	1760	1704	-3,18%	179.art	1720	1722	0,12%
183.equake	447	444	-0,67%	183.equake	443	443	0,00%
187.facerec	390	393	0,77%	187.facerec	394	394	0,00%
188.amp	669	680	1,64%	188.amp	662	698	5,44%
189.lucas	854	832	-2,58%	189.lucas	872	829	-4,93%
191.fma3d	273	273	0,00%	191.fma3d	272	271	-0,37%
200.sixtrack	182	184	1,10%	200.sixtrack	184	186	1,09%
301.apsi	507	515	1,58%	301.apsi	510	543	6,47%
GeoMean	522,64	529,53	1,32%	GeoMean	523,994	543,123	3,65%

No speculation support in both schedulers

Control speculation support in both schedulers

Gains on selected SPEC tests

Biggest gains come from:

- Pipelining + speculation + renaming
 - We can always rename floating-point registers
- Backend tuning
 - Especially placement of stop bits

171.swim, +5,05%

172.mgrid, +12,35%

178.galgel, +18,66%

301.apsi, +6,47%

Propagating alias information to RTL

- Implemented in 2005
 - Propagates flow-insensitive points-to sets
 - Generates better alias set numbers based on a high-level points-to information
 - Creates a mapping between RTL MEMs and original trees
- The numbers of new disambiguations are not convincing
 - Saved information is not precise enough?
 - Alias analysis could not be run after iv optimizations
 - We lose something important when saving?
 - Flow sensitivity, tree \leftrightarrow RTL mapping
 - RTL optimizations destroy saved info?

Ideas for aliasing work in 2007

- Update the patch for current trunk
- Recalculate the numbers
 - Latest improvements to the aliaser might help
- Do not (again) coalesce SSA pointers with different points-to sets
 - Must be updated for the new out-of-ssa pass
- Support TARGET_MEM_REF trees in the aliaser
- Implement the verifier of the saved information to catch the inconsistencies introduced on RTL
- Experiment with “base+offset” tracking aliasing pass on RTL
 - Good for Itanium that doesn't have b+o addressing

Improving modulo scheduling in GCC

- Make the modulo scheduling implementation to work on Itanium
 - Need to fix some code generation bugs
 - First patch is already published
- Improve data dependency analysis on RTL
 - Propagate data dependence info from Tree SSA to RTL
 - Implement the verifier to check the saved data
 - Fix the problems found via the verifier on RTL
- Use the saved data in modulo scheduling
 - Test the implementation on small tests and SPEC
- The project is sponsored by Intel Russia
 - Started in April 2007

Questions?